2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC

Host Sponsor UNC Charlotte

ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION

Paper ID #51241

How LLM-Based Agents Shape Engineering Design Thinking: A Qualitative Study with Undergraduate Software Engineering Students

Liuying Gong, Zhejiang University

Liuying Gong received the bachelor's degree in engineering from Zhejiang University. She is currently pursuing the Ph.D. degree with the School of Public Affairs, Zhejiang University, Hangzhou, China. Her mentors are Prof. Y. Min. Her current research interests include AI for education, engineering education, and AI talent cultivation.

How LLM-Based Agents Shape Engineering Design Thinking: A Qualitative Study with Undergraduate Software Engineering Students

Abstract

The integration of Large Language Model-based agents (LLM-based agents), such as GitHub Copilot and ChatGPT, into software engineering practices is advancing at an unprecedented rate. These tools have evolved beyond their original role as information retrieval systems to become active "partners." This shift has sparked urgent discussions within the field of higher education. While existing research has predominantly focused on the quantitative assessment of the impact of LLM-based agents on coding efficiency and accuracy, their influence on students' engineering design thinking remains insufficiently explored. Design is the core of engineering practice. Engineering design thinking, which involves skills such as systematic analysis, abstract modeling, trade-off evaluation, and creative problem-solving, serves as one of the foundations of engineering education. Therefore, investigating how LLM-based agents shape students' engineering design thinking is crucial for understanding and guiding the future trajectory of engineering education.

This study employed a qualitative approach within a semester-long software engineering course at a leading research university. Forty undergraduate students, organized into eight teams, were recruited to develop complex AI-native systems while interacting with LLM-based agents. Data were collected through three methods: (1) post-task semi-structured interviews, (2) interaction logs between students and LLM-based agents, and (3) final software artifacts, including the documentations.

The findings reveal that LLM-based agents took on significant responsibilities for information retrieval and code generation, allowing students to focus more on architectural design. This led to a shift in engineering design thinking process, moving from "ideate-prototype-test" to a rapid "prompt-evaluate-refine" loop. Distinct design strategies emerged. Some students effectively used the agent's outputs as inspiration, while others exhibited a tendency to "delegate". Furthermore, cognitive load was redistributed from syntactic implementation to upstream problem scoping and downstream evaluation. The study's primary contribution lies in providing an empirical foundation and strategic framework for this essential transformation of engineering education.

1 Introduction

The integration of Large Language Models-based agents (LLM-based agents) into software engineering practice is advancing at an unprecedented speed, fundamentally altering the landscape of how code is produced and problems are solved. Tools have evolved far beyond their initial roles as advanced information retrieval systems. While early implementations like GitHub Copilot served as intelligent autocomplete plugins [1], [2], a new generation of Artificial Intelligence (AI) native Integrated Development Environments (IDEs), epitomized by Cursor [3], is emerging. These platforms function as autonomous agents capable of perceiving entire codebases, executing multi-file refactoring, and proactively debugging [4]. Industry reports indicate that a significant portion of modern code is now not merely augmented but architected by LLM-based agents, leading to substantial shifts in productivity standards and redefining the threshold of entry-level competency [5]. In the context of higher education, this technological disruption has sparked an urgent debate regarding curriculum adaptation. While students are rapidly adopting these agents, the educational community is still grappling with understanding the nature of this human-agent partnership and its long-term implications for professional competency [6].

Despite the widespread adoption of these tools, current educational research has predominantly focused on quantitative metrics of coding efficiency and syntactical accuracy, leaving the deeper cognitive impact on engineering design thinking largely unexplored. Existing studies often evaluate LLM-based agents based on the correctness of generated solutions or the reduction in time-to-completion for specific programming tasks [7], [8]. While these metrics are valuable, they fail to capture the holistic nature of engineering design. Engineering design is not merely the act of writing code; it is a systematic, intelligent process in which designers generate, evaluate, and specify concepts for devices, systems, or processes whose form and function achieve clients' objectives or users' needs while satisfying a specified set of constraints [9]. There is a paucity of empirical evidence detailing how the presence of an intelligent agent influences designers' higher-order thinking skills, specifically in how students conceptualize architectures and navigate design trade-offs.

Engineering design thinking represents the cornerstone of engineering education, and its preservation and evolution are critical in the AI era. Scholars have long argued that the essence of engineering lies in the iterative process of defining problems and synthesizing solutions, rather than just the final artifact [9], [10], [11]. As LLM-based agents increasingly handle implementation tasks, engineering education must shift its focus toward the higher-order thinking skills required to guide these agents. However, it is currently unclear whether students are making this transition effectively. They may be using AI for inspiration and rapid prototyping, or they may be adopting a "delegation" mindset by offloading critical thinking and design judgment to the models [12], [13]. Understanding this behavioral dichotomy is essential for educators who aim to cultivate engineers capable of critically mastering AI tools rather than being passively led by

them.

To address this critical gap, this study employs a qualitative multi-case approach to investigate how undergraduate students reshape their engineering design thinking while interacting with LLM-based agents. In contrast to output-oriented quantitative studies, this research focuses more on the process. We analyzed data from 40 students at a research university, using semi-structured interviews, interaction logs, and design outcomes to identify shifting patterns in design thinking. By documenting these emergent behaviors and strategies, this study provides a much-needed empirical foundation for re-evaluating engineering design pedagogy, moving beyond simple policies of prohibition or endorsement toward a framework that redefines engineering design thinking for the age of AI.

2 Literature Review

2.1 Engineering Design Thinking

Engineering design is widely recognized not as a linear trajectory but as a complex, iterative cognitive process. Dym et al. defined engineering design thinking as a complex cognitive activity that involves inquiry, questioning, and diverging-converging thinking patterns [9]. Central to this definition is the ability to tolerate ambiguity and navigate "ill-structured" problems, which lack a single correct solution and require decision-making under uncertainty [14].

Traditional design thinking processes emphasize the "Ideate-Prototype-Test" cycle, where students represent ideas through sketching or low-fidelity prototyping to solicit feedback and refine constraints [15]. In the context of software design, this involves architectural decision-making, requirement analysis, and trade-off evaluation, distinct from mere syntactic implementation [16].

To translate the engineering design thinking processes into measurable educational outcomes, researchers have developed rigorous quantitative frameworks to assess student design behaviors and competencies. The most prominent method for analyzing the process of design is the Timeline Analysis Method pioneered by Atman et al. [14]. Through extensive empirical studies tracking how engineering students allocate time across different design stages, they conceptualized the design process as a sequence of transitions between states (e.g., Problem Scoping, Information Gathering, Modeling). Their quantitative findings established a critical baseline: expert designers and high-performing students spend a significantly larger proportion of their time in "problem scoping" to gather information and question constraints before attempting to build a solution. In contrast, novices tend to exhibit a "premature fixation" on initial ideas, rushing immediately into implementation [17], [18].

2.2 LLM-based Agents in Software Engineering Education

Agents or autonomous agents are defined as systems capable of perceiving their environment and making decisions to achieve specific objectives based on those perceptions [19], [20]. Early iterations of agents lacked adaptability and generalization, creating a distinct gap from human-level intelligence [21]. However, the recent emergence of LLMs has bridged this gap, enhancing agents' capabilities in command interpretation, knowledge acquisition, and human-like reasoning [22], [23]. Unlike traditional software tools, LLM-based agents utilize the LLM as a primary decision-making engine or "core controller" augmented by critical human-like characteristics such as planning, memory, and tool use [7], [24], [25].

In the context of software engineering education, this technological leap signifies a transition from "tools" to "partners." While early implementations like standard code completers functioned as advanced information retrieval systems, modern platforms (e.g., Cursor, AutoGPT) exemplify the agentic architecture described above. They do not merely suggest syntax but perceive the entire codebase (Environment), plan refactoring steps (Planning), and execute changes across multiple files (Action) [26].

Despite this profound shift, current educational research has largely remained focused on the output of these systems rather than the interaction. Studies have predominantly adopted a quantitative lens, demonstrating, for instance, that students using LLM-based agents complete coding tasks significantly faster and with higher assessment scores [27]. However, there is a paucity of empirical evidence detailing how the presence of an agent, which is capable of acting as a "core controller" for software design process, affects how students navigate the ambiguity of engineering design itself.

3 Methodology

3.1 Research Method

To investigate the shifts in engineering design thinking facilitated by LLM-based agents, this study employs an exploratory approach of a primarily qualitative and interpretative nature [28]. Given that the integration of LLM-based agents into design workflows is a contemporary and context-dependent phenomenon, a qualitative inquiry is essential to capture the how and why behind student behaviors, rather than merely measuring the output metrics. This design allows for an in-depth examination of the design process within its real-life context, capturing the dynamic and often non-linear interactions between the student and LLM-based agents [29]. Specifically, we adopted a sentence-level unit of analysis to examine the qualitative data collected through interviews and supplementary channels. Leveraging interdisciplinary expertise at the intersection of software engineering and engineering education, we inductively synthesized emerging trends in design thinking transformations from the bottom up.

3.2 Case Selection

The study was conducted at Zhejiang University, a top-tier research university in China recognized for its excellence in engineering education, involving 40 undergraduate students majoring in Software Engineering. To ensure the research focused on high-level design thinking rather than remedial syntax debugging, we employed a purposive sampling strategy based on threshold competency [30]. This required that all participants had successfully completed their sophomore core curriculum, including Data Structures, Databases, and Fundamentals of Software Engineering. Focusing a homogenous group of emerging practitioners was necessary to ensure that interactions with the LLM-based agents reflected high-level design processes rather than simple debugging.

The study was integrated into a semester-long course. The 40 participants were organized into eight teams via a process of free grouping with mandatory role requirements, ensuring each group established a clear division of labor with distinct positions such as Product Manager and Chief Technology Officer. Each team was tasked with the design and implementation of a complex, AI-embedded system over a ten-week period. These projects were selected for their close alignment with industrial needs, serving as authentic "ill-structured problems" which lack a single correct solution and require high-level abstraction under conflicting constraints [31]. Specific project topics included "the AI-driven enterprise financial reimbursement system", "intelligent learning assistant", and "LLM-based platform for automated industry chain mapping." Unlike closed-ended coding exercises, these tasks contained intentional ambiguity, requiring teams to navigate the full development lifecycle from problem definition to system deployment. Throughout this process, students utilized LLM-based agents to assist in building these AI-native applications, mirroring professional engineering workflows.

3.3 Data Collection

We employed a methodological triangulation strategy to ensure the validity and robustness of the findings [32]. Data were collected through three sources: interaction logs covering the full workflow, post-course semi-structured interviews with group leaders, and final design artifacts such as codebases and documents. It is important to note that the interviews served as the primary source of analysis, while interaction logs and design artifacts (see **Figure 1**) providing context for cross-verification. The interview protocol (see **Table 1**) specifically examined the utility of LLM-based agents across various phases of software design, the nature of human-AI interaction, and the subsequent impacts of these interactions on both team dynamics and individual cognition.

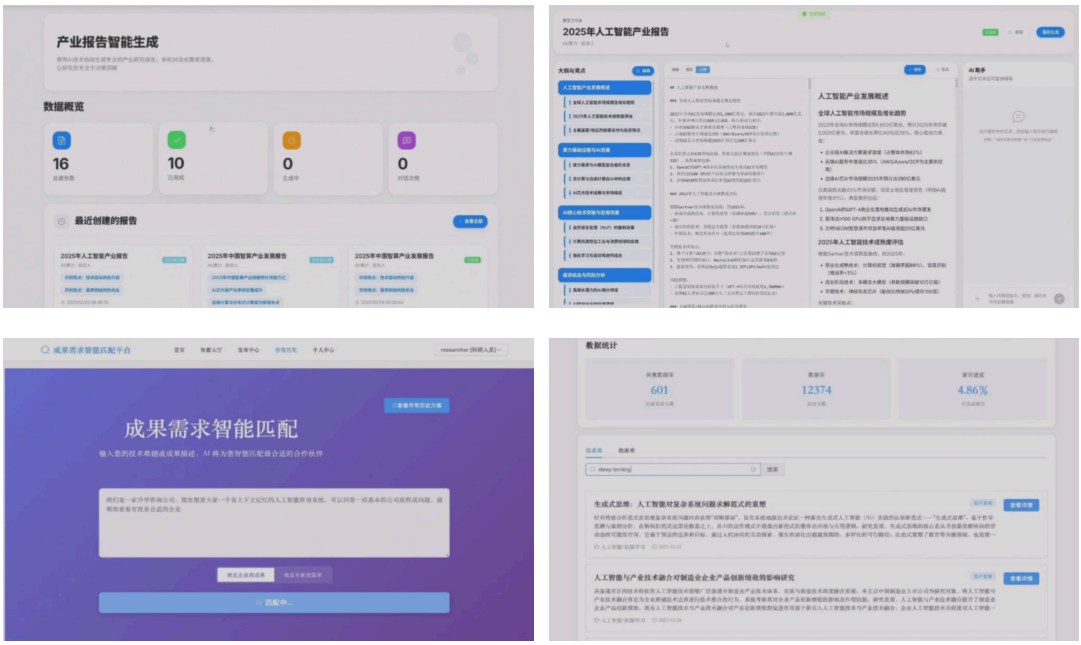

Figure 1 Screenshots of final software artifacts from some groups.

Table 1 Semi-structured Interview Protocol for this study

Category	**ID**	**Core Inquiry & Probing Questions**
A. Background & Context	**Q1**	**Project Overview & Role** Please briefly describe your group's project and define your primary role within the group.
	Q2	**Tool Utilization Profile** Which LLM-based agents did you utilize (e.g., ChatGPT, Copilot, Claude, Gemini, Cursor)?
	Q3	**Role Conceptualization** If the LLM-based agent were a new member of your group, how would you define its role? Why?
B. Requirement & Design Phase	**Q4**	**Requirement Analysis Strategy** How was the LLM-based agent utilized during the initial requirement analysis? *Probe 1: Did it foster divergent thinking (expanding feature scope) or convergent thinking (pinpointing core points)?* *Probe 2: Did it introduce unexpected perspectives that altered your understanding of user needs?*
	Q5	**Architectural Design Contribution** What specific role did the LLM-based agents play in system architecture design? Please provide a concrete example.
	Q6	**Methodological Shift** Do you lean more towards a Top-Down approach (LLM-based agents generates the framework first) or a Bottom-Up approach (writing modules first, then assembling)?
C.	**Q7**	**Technical Decision Support**

Implementation & Decision Making		How did the LLM-based agents assist in selecting technical solutions? *Probe 1: How did you verify the feasibility of the suggestions?* *Probe 2: Did you ever reject the solutions? If so, why?*
	Q8	**Problem Solving & Debugging** How did the LLM-based agents perform when encountering technical bottlenecks?
D. Collaboration & Workflow	**Q9**	**Impact on Division of Labor** Did the presence of LLM-based agents alter the collaboration model?
	Q10	**Cognitive Load Distribution** Overall, in which phase did your group focus the most time and cognitive load? (e.g., Requirement alignment, prototyping, coding, prompt engineering, or testing/debugging?)
E. Reflexivity & Future Implications	**Q11**	**Recursive Understanding** Since your system itself embeds AI features, did you experience any "recursive" insights? *Probe: When designing prompts for your end-users, did you draw upon your own experience as a developer prompting the agents?*
	Q12	**Impact on Design Thinking** Do you feel that over-reliance on LLM-based agents weaken or augment your specific capabilities like engineering design thinking?
	Q13	**Dependency Assessment** Without LLM-based agents' assistance, do you believe you could still independently complete a system design of this complexity? Why or why not?
	Q14	**General Feedback** Beyond this specific course, do you have other insights or confusions regarding the use of LLM-based agents in software engineering?

4 Results

The thematic analysis of the triangulated data revealed profound shifts in how engineering students approach design tasks when assisted by LLM-based agents. These findings are categorized into three major themes: the transformation of the iterative engineering design thinking process, the emergence of distinct interaction strategies, and the redistribution of cognitive load.

4.1 The Transition to the "Prompt-Evaluate-Refine" Loop

The most significant finding derived from the thematic analysis is the evolution of the engineering design thinking process from the traditional "Ideate-Prototype-Test" cycle to a "Prompt-Evaluate-Refine" (PER) loop (see **Figure 2**). It is crucial to note that this transition does not imply a complete negation of the established workflow. Instead, it represents a multidimensional enhancement of each stage, serving as a strategic

response to the embedding of intelligent technologies in engineering tasks.

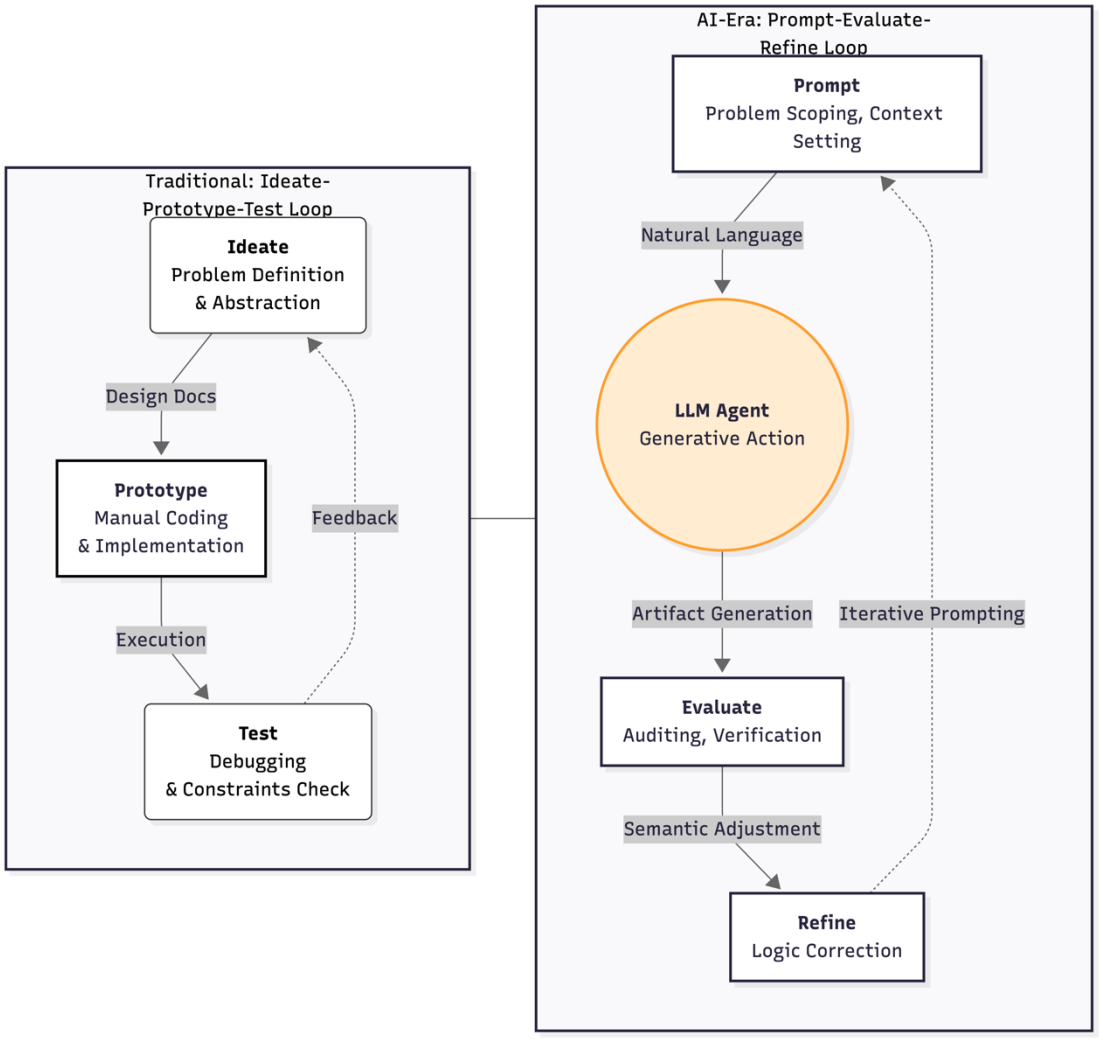

Figure 2 The Transformation of Engineering Design Thinking. Left: the traditional Ideate-Prototype-Test cycle; Right: the emerging Prompt-Evaluate-Refine loop.

"Prompting" in this context evolves beyond its literal meaning of issuing instructions to become the primary vehicle for Ideation and Problem Scoping. In the software design setting, prompting acts as a mechanism for domain expansion and architectural framing, especially by allowing the agents to play specific roles. As the leader of the "Smart Finance" project (Participant P03) articulated, prompting served to bridge the gap between limited student experience and industrial reality:

> *As students, we had either zero or very limited experience with reimbursement processes. The LLM helped us enumerate many real-world enterprise scenarios, such as multi-currency conversion, invoice verification, and compliance checks for travel standards, effectively filling our domain knowledge gaps (Participant P03).*

By interacting with the agent, the team expanded their initial scope from basic

functional requirements to complex enterprise-level perspectives. Thus, prompting becomes a form of generative ideation where system boundaries and architectural interfaces are defined.

"Evaluating" assumes the critical function previously held by Prototyping and early-stage Testing. Here, it is defined as the critical auditing of agent-generated artifacts, requiring students to shift their role from code programmers to technical reviewers. Participants frequently conceptualized the LLM-based agent as a "knowledgeable but judgment-deficient intern," underscoring the necessity of human oversight. They noted that dismissing LLM-based agents' suggestions after evaluation was common, especially in the early phases. The models frequently proposed solutions that demonstrated a fundamental misalignment with real-world constraints. Evaluating, therefore, becomes a mentorship process where the human engineer must filter hallucinations and ensure the proposed solution aligns with the specific project constraints.

"Refining" redefines the Iteration phase, transforming it from a process of code debugging to semantic adjustment. In the PER loop, refining refers to the iterative adjustment of the prompt logic based on the evaluation results. This shifts the nature of problem-solving from trial-and-error coding to semantic debugging. When encountering technical bottlenecks, students reported that they no longer focused on syntax correction but on formulating the "right question" to guide the agent. Participant P02 noted during the interview:

> *If you can't prompt the answer, it means your understanding of the bug isn't deep enough (Participant P02).*

This process turns debugging into an exercise in logical articulation, where the improvement of the system relies on the precision of the natural language description provided to the model.

The fundamental characteristic of the PER loop, compared to its predecessor, lies in its generative breadth and temporal agility. This mode enables development teams to identify viable design alternatives more rapidly, as the cost of generating a prototype is largely reduced. Furthermore, due to the interactive nature of LLM-based agents, the design thinking process becomes continuous and agile. Students can cycle through the PER loops multiple times in the span of minutes, allowing for a dynamic exploration of the solution space that was previously constrained by the labor-intensive nature of manual construction.

4.2 The Emergence of Distinct Interaction Strategies

Building upon the transition to the Prompt-Evaluate-Refine loop, our interaction logs and interviews revealed that students did not navigate this loop uniformly. Instead, two distinct interaction strategies emerged, distinguishing how students leveraged the

generative capabilities of LLM-based agents to approach engineering tasks. While some students utilized the technology to amplify their cognitive capacity through inspiration, the other exhibited a tendency to bypass cognitive effort through delegation. These divergent strategies fundamentally dictated the quality and depth of the final engineering designs.

A significant portion of participants approached the agent primarily as a cognitive scaffold. This inspiration-driven collaboration was characterized by the formulation of a preliminary design prior to engaging the LLM-based agents. These students maintained strict control over the system intent, using the agent to generate alternatives rather than asking it to dictate the final logic. The interaction logs show instances of informed decision-making where students actively evaluated the suggestions.

> *I used the agent to brainstorm. I asked it for different ways to handle the concurrency issue, and used its code as inspiration but rewrote the core logic to fit my specific constraints (Participant P08).*

> *First, I provided the LLM-based agents with our project requirements and module breakdown to get a preliminary database design. Then, I iteratively modified it and repeatedly asked the agents for feedback, checking if the changes were appropriate or if there were any loopholes. I continued refining the design until no obvious flaws remained before adopting it (Participant P02).*

In contrast, the remaining participants exhibited a clear tendency toward delegation-driven dependency, often engaging in extensive cognitive offloading. These students accepted the output of the agent as the definitive truth, frequently using vague prompts such as “write the whole program” which resulted in a decoupling of the final artifact from their actual understanding. Participant P23’s experience typifies this passive mindset:

> *When the test failed, I just pasted the error message back into the chat and hoped it would fix itself. I was basically just passing messages (Participant P23).*

This behavior mirrors the recently popularized concept of “Vibe Coding” [33], where development is driven by natural language intuition and the “vibe” of the output rather than rigorous syntactic construction. However, while vibe coding for experts implies a high-level orchestration backed by the ability to verify, for these students, it manifested as a reliance. They adopted the form of vibe coding but lacked the technical judgment regarding the results. Consequently, their “Refine” phase often degenerated into code churn or prompt churn, where the lack of foundational understanding turned the coding process into a game of probability rather than engineering.

The interaction strategies have profound implications for the overall quality of software

engineering outputs. We observed that the presence of LLM-based agents has significantly raised the performance floor. It is worth noting that the group to which Participant P23 belonged did not perform well in the mid-term evaluation and was lagging behind in progress. Ultimately, however, they still presented a complete and passable project. The adoption of a "Vibe Coding" style allows even novices to produce functional, complex systems that would previously require weeks of effort. However, the determinant of the performance ceiling has shifted. Excellence is no longer defined merely by code completeness, but by the rigor of evaluation and the depth of productization. The highest-quality artifacts in our study were not characterized by the speed of their generation, but by their alignment with real-world needs, such as systems co-developed with university finance departments.

4.3 The Redistribution of Cognitive Load

The integration of LLM-based agents significantly altered the distribution of cognitive load for both individual students and the project teams. Our study indicates that while the burden of writing code syntax decreased, the cognitive load shifted toward other critical aspects of the software engineering design thinking.

At the individual level, participants reported their focuses was redirected to two specific areas: defining precise requirements before coding, and auditing the output after generating. This change required students to have a clearer understanding of the system logic upfront and a higher vigilance during testing. As Participant P37 noted during the interview:

> *I was exhausted by the end because I spent most of my time designing complex prompts before the code existed, and then checking the output for subtle logic errors (Participant P37).*

At the team level, our study revealed a new cognitive challenge regarding the management of shared context, leading to a restructuring of the division of labor. Participants pointed out that interactions with LLM-based agents, specifically the history of prompts and constraints, are often private. If one member generated the backend logic using a specific conversation history, it was difficult for a separate frontend developer to seamlessly align with that hidden context.

To mitigate this coordination cost, teams shifted from the traditional horizontal separation of duties such as frontend and backend, to a feature-based division. Since the LLM-based agents empowered individual students to handle full-stack development, teams increasingly assigned entire functional modules to single members. This structural change minimized the need to synchronize complex contexts between different developers, as the entire logic from database to UI was maintained within a single, coherent user-agent conversation. Participant P18 explained:

> *We tried different ways of splitting the work. After finalizing the technical stack, architecture, and APIs, we decided to divide tasks by functional features. One person owns a whole feature. This way, we almost don't have to explain the context to anyone else, avoiding the mess of trying to align two different chat histories (Participant P18).*

5 Discussion

5.1 Redefining Design Thinking in the AI Era

The findings of this study extend existing theoretical frameworks of engineering design thinking to accommodate the integration of LLM-based agents. Traditional models, such as those proposed by Dym et al. [9], characterize design as a cognitive cycle of "divergent inquiry" and "convergent synthesis." Our observations of the "Prompt-Evaluate-Refine" loop suggest that in the AI era, these core cognitive processes are being augmented through interaction with intelligent agents. Specifically, the phase of divergent inquiry is now enriched by prompting, which utilizes the agents to explore options that extend beyond the student's immediate knowledge base. Similarly, convergent synthesis evolves from a purely manual construction task into a process of evaluation, where students integrate AI-generated outputs into a coherent system.

Therefore, we propose that the definition of engineering design thinking should be broadened. This encompasses the ability to decompose ill-structured problems into promptable modules and the rigorous judgment required to validate outputs against engineering constraints. More importantly, the LLM-based agent is not an infinite resource but a constrained component within the design environment. Just as engineers must account for physical limitations like material strength or processing power, they must now factor in the sociotechnical constraints of the agent, such as its limited context window, lack of persistent memory across sessions, and stochastic nature. Thus, modern engineering design thinking requires the ability to model the agents as part of the system's environment, actively designing workflows that mitigate the agent's inherent limitations to ensure the robustness of the final solution.

5.2 From Prohibition to Critical Mastery

The emergence of the "Delegation" strategy observed in our study highlights a significant risk that students can now generate functional artifacts without deep understanding. This calls for a paradigm shift in engineering pedagogy, moving away from simple "ban or ignore" policies toward a framework that fosters critical mastery.

First, assessment methods must evolve from checking the final code to evaluating the design process. Since high-quality code can be generated effortlessly, the final artifact is no longer a sufficient proxy for competence. Educators should place greater weight on how students arrived at the solution. This could involve assessing the interaction logs as part of the submission, grading students on the quality of their prompts and,

more importantly, their rationale for accepting or rejecting the agents' suggestions.

Second, curricula must explicitly teach "Skeptical Collaboration" with LLM-based agents. Given the tendency to blindly trust the LLM-based agents, courses should incorporate exercises specifically designed to break this trust. For example, students could be tasked with auditing flawed AI-generated code to identify subtle security vulnerabilities or architectural defects. This trains students to treat the LLM-based agent not as an oracle, but as a fallible assistant that requires constant supervision.

6 Conclusion

This study set out to explore the qualitative impact of LLM-based agents on the engineering design thinking of undergraduate students. The thematic analysis of the triangulated data confirms that the integration of LLM-based agents does not merely enhance efficiency, it fundamentally reconfigures the cognitive mechanisms of engineering practice.

This reconfiguration is evidenced by three shifts. First, the study identifies a structural evolution from the traditional "Ideate-Prototype-Test" cycle to the rapid, conversational PER loop. This transition signifies that the primary cognitive activity in engineering is shifting from manual construction to architectural orchestration. Second, the behavioral divergence between "Inspiration-Driven Collaboration" and "Delegation-Driven Dependency" highlights a critical educational challenge. There is a risk that students may adopt a passive acceptance of AI outputs, decoupling the final artifact from their own understanding. Third, mental effort has migrated away from the middle stage of implementation toward the upstream phase of requirement definition and the downstream phase of evaluation. Additionally, at the team level, this redistribution introduces a new "context management" overhead, where the ability to synchronize interaction histories and system prompts has become as critical as managing the code itself.

Building upon these findings, our study offers recommendations for both theoretical research and pedagogical practice, aiming to provide direction for the evolution of engineering design education. While this study was conducted within a high-performing cohort, we posit that the identified tension between inspiration and delegation is transferable to broader educational contexts. In settings where students possess less foundational expertise, the risk of "Delegation-Driven Dependency" may be even more pronounced due to a reduced capacity for critical evaluation, making the explicit cultivation of these AI literacy skills universally urgent. We advocate for moving beyond simple policies of prohibition or endorsement, steering instead toward a framework that redefines engineering design thinking for the age of AI.

References

[1] A. Moradi Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, and Z. M. (Jack) Jiang, 'GitHub Copilot AI pair programmer: Asset or Liability?', *Journal of Systems and Software*, vol. 203, p. 111734, Sept. 2023, doi: 10.1016/j.jss.2023.111734.

[2] S. Barke, M. B. James, and N. Polikarpova, 'Grounded Copilot: How Programmers Interact with Code-Generating Models', *Proc. ACM Program. Lang.*, vol. 7, no. OOPSLA1, pp. 85–111, Apr. 2023, doi: 10.1145/3586030.

[3] 'Cursor Docs', Cursor Documentation. Accessed: Dec. 16, 2025. [Online]. Available: https://cursor.com/docs

[4] J. Yang *et al.*, 'SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering', Nov. 11, 2024, *arXiv*: arXiv:2405.15793. doi: 10.48550/arXiv.2405.15793.

[5] A. Ziegler *et al.*, 'Measuring GitHub Copilot's Impact on Productivity', *Commun. ACM*, vol. 67, no. 3, pp. 54–63, Mar. 2024, doi: 10.1145/3633453.

[6] J. Prather *et al.*, '"It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers', *ACM Trans. Comput.-Hum. Interact.*, vol. 31, no. 1, pp. 1–31, Feb. 2024, doi: 10.1145/3617367.

[7] L. Wang *et al.*, 'A survey on large language model based autonomous agents', *Front. Comput. Sci.*, vol. 18, no. 6, p. 186345, Mar. 2024, doi: 10.1007/s11704-024-40231-1.

[8] M. Rodriguez, 'Research: Quantifying GitHub Copilot's impact on code quality', The GitHub Blog. Accessed: Dec. 16, 2025. [Online]. Available: https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-code-quality/

[9] C. L. Dym, A. M. Agogino, O. Eris, D. D. Frey, and L. J. Leifer, 'Engineering Design Thinking, Teaching, and Learning', *Journal of Engineering Education*, vol. 94, no. 1, pp. 103–120, Jan. 2005, doi: 10.1002/j.2168-9830.2005.tb00832.x.

[10] D. A. Schön, *The Reflective Practitioner: How Professionals Think in Action*. London: Routledge, 2017. doi: 10.4324/9781315237473.

[11] H. A. Simon, *The Sciences of the Artificial*. Accessed: Dec. 17, 2025. [Online]. Available: https://direct.mit.edu/books/monograph/4551/The-Sciences-of-the-Artificial

[12] E. F. Risko and S. J. Gilbert, 'Cognitive Offloading', *Trends in Cognitive Sciences*, vol. 20, no. 9, pp. 676–688, Sept. 2016, doi: 10.1016/j.tics.2016.07.002.

[13] F. Dell'Acqua *et al.*, 'Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality', *SSRN Journal*, 2023, doi: 10.2139/ssrn.4573321.

[14] C. J. Atman, R. S. Adams, M. E. Cardella, J. Turns, S. Mosborg, and J. Saleem, 'Engineering Design Processes: A Comparison of Students and Expert Practitioners', *J of Engineering Edu*, vol. 96, no. 4, pp. 359–379, Oct. 2007, doi: 10.1002/j.2168-9830.2007.tb00945.x.

[15] T. Brown, 'Design thinking', *Harv Bus Rev*, vol. 86, no. 6, pp. 84–92, 141, June 2008.

[16] A. Von Mayrhauser and A. M. Vans, 'Program comprehension during software maintenance and evolution', *Computer*, vol. 28, no. 8, pp. 44–55, Aug. 1995, doi: 10.1109/2.402076.

[17] C. J. Atman, J. R. Chimka, K. M. Bursic, and H. L. Nachtmann, 'A comparison of freshman and senior engineering design processes', *Design Studies*, vol. 20, no. 2, pp. 131–152, Mar. 1999, doi: 10.1016/S0142-694X(98)00031-3.

[18] M. C. Yang, 'A study of prototypes, design activity, and design outcome', *Design Studies*, vol. 26, no. 6, pp. 649–669, Nov. 2005, doi: 10.1016/j.destud.2005.04.005.

[19] M. J. Wooldridge and N. R. Jennings, 'Intelligent Agents: Theory and Practice', *The Knowledge Engineering Review*, vol. 10, no. 2, Art. no. 2, 1995, doi: 10.1017/S0269888900008122.

[20] S. Franklin and A. Graesser, 'Is It an agent, or just a program?: A taxonomy for autonomous agents', in *Intelligent Agents III Agent Theories, Architectures, and Languages*, J. P. Müller, M. J. Wooldridge, and N. R. Jennings, Eds, Berlin, Heidelberg: Springer, 1997, pp. 21–35. doi: 10.1007/BFb0013570.

[21] X. Wang and H. Su, 'Completely model-free RL-based consensus of continuous-time multi-agent systems', *Applied Mathematics and Computation*, vol. 382, p. 125312, Oct. 2020, doi: 10.1016/j.amc.2020.125312.

[22] Z. Zhang, M. Fang, L. Chen, M.-R. Namazi-Rad, and J. Wang, 'How Do Large Language Models Capture the Ever-changing World Knowledge? A Review of Recent Advances', in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds, Singapore: A Systematic

Investigation of Commonsense Knowledge, Dec. 2023, pp. 8289–8311. doi: 10.18653/v1/2023.emnlp-main.516.

[23] L. Yang *et al.*, 'Supervised Knowledge Makes Large Language Models Better In-context Learners', 2023, doi: 10.48550/ARXIV.2312.15918.

[24] T. Schick *et al.*, 'Toolformer: Language Models Can Teach Themselves to Use Tools', *Advances in Neural Information Processing Systems*, vol. 36, pp. 68539–68551, Dec. 2023.

[25] Z. Xi *et al.*, 'The rise and potential of large language model based agents: a survey', *Sci. China Inf. Sci.*, vol. 68, no. 2, p. 121101, Jan. 2025, doi: 10.1007/s11432-024-4222-0.

[26] H. Mozannar, G. Bansal, A. Fourney, and E. Horvitz, 'Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming', in *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, in CHI '24. New York, NY, USA: Association for Computing Machinery, May 2024, pp. 1–16. doi: 10.1145/3613904.3641936.

[27] M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, and T. Grossman, 'Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming', in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, in CHI '23. New York, NY, USA: Association for Computing Machinery, Apr. 2023, pp. 1–23. doi: 10.1145/3544548.3580919.

[28] R. K. Yin, *Case study research and applications: design and methods*, Sixth edition. Los Angeles London New Delhi Singapore Washington DC Melbourne: SAGE, 2018.

[29] J. W. Creswell and C. N. Poth, *Qualitative inquiry and research design*, Fourth edition. Los Angeles: SAGE, 2018.

[30] M. Q. Patton, *Qualitative research & evaluation methods: integrating theory and practice*, Fourth edition. Los Angeles London New Delhi Singapore Washington DC: SAGE, 2015.

[31] D. H. Jonassen, 'Toward a design theory of problem solving', *ETR&D*, vol. 48, no. 4, pp. 63–85, Dec. 2000, doi: 10.1007/BF02300500.

[32] N. K. Denzin, *The Research Act: A Theoretical Introduction to Sociological Methods*. New York: Routledge, 2017. doi: 10.4324/9781315134543.

[33] Y. Ge *et al.*, 'A Survey of Vibe Coding with Large Language Models', Dec. 21, 2025, *arXiv*: arXiv:2510.12399. doi: 10.48550/arXiv.2510.12399.